EECS 280 Final Exam
Fall 2019
Reference material

Please do not open this booklet before the exam begins.

This booklet contains reference material that you will need in working some of the sections in the exam.

You will need to turn this in with your exam.

You may write on these pages but nothing you write here will be considered in grading your exam.

All your answers must be made in the exam booklet.

This page intentionally left blank.
You may use this space for scratch work, but we will not grade anything here.

Section 1 Reference Sheet

Section 1b of the exam refers to the following code.

```
#include <iostream>

using namespace std;

class Fruit {
      public:
            Fruit(){
                  cout << "fruit ctor" << endl;
            }

            Fruit (const Fruit & other) {
                 cout << "fruit copy ctor" << endl;
            }

            virtual ~Fruit(){
                  cout << "fruit dtor" << endl;
            }
};

class Apple : public Fruit {
      public:
            Apple(){
                  cout << "apple ctor" << endl;
            }

            Apple (const Apple & other) {
                 cout << "apple copy ctor" << endl;
            }

            ~Apple(){
                  cout << "apple dtor" << endl;
            }
};
```

Section 1c of the exam refers to the following code.

```
class TripException {};
class FlightException {};
class OtherException {};

void bookFlight(int num_persons, const string &src, const string &dst){
    if ( num_persons > 15 ) { throw TripException(); }
    if ( src == dst  ) { throw FlightException(); }
}

void makeTrip(int persons, const string &src, const string &dst ) {
    if ( persons <= 0 ) { throw OtherException();}
    int count = 0;
    try {
         bookFlight(persons,src, dst);
    }
    catch (const FlightException &e) {
         cout << "Source and Destination are identical" << endl;
          throw OtherException();
    }
    catch (const OtherException) {
        cout << "Error encountered." << endl;
   }

    cout << "Booking finished!" << endl;
}

int main() {
    int num_people;
    string source;
    string dest;
    cin >> num_people >> source >> dest;
    try {
         makeTrip(num_people, source, dest);
         cout << "After makeTrip()" << endl;
    }
    catch (const TripException &e) {
        cout << "We can't book your trip." << endl;
    }
    catch (...) {
        cout << "Invalid input." << endl;
   }
}
```

Section 2 Reference Sheet

Consider the following code:

```
class Directory {
private:
      string dir_name;  // name of the directory
      int num_files;  // actual number of files in the directory
      int capacity; // maximum possible number of files in the directory
      File* file_arr;   // a dynamically sized array of files

public:
      // Takes in a vector of files and constructs a directory of those
      // files
      // REQUIRES: The vector size is <= 100.
      Directory(const string &dir_name_in, const vector<File> &files_in);
            //*You will implement this in part (2a)*

      // EFFECTS: Performs clean-up of memory of the Directory object
      ~Directory();
            //*You will implement this function in part (2d)*

      // EFFECTS: Performs a deep copy from other into this Directory
      Directory &operator=(const Directory & other);
            //*This function is implemented in part (2c)*

      // EFFECTS: Resizes the directory so the array of files is at minimal
      // possible size to hold all data
      // The new capacity of the directory is equal to the
      // number of files in the directory
      // Appends "_compressed" to the directory name
      void shrink() {
            //*This function is implemented in part (2b)*
      }

};

class File {
private:
      string file_name;
public:
      File(string name_in):file_name(name_in) {}
      File() {}
};
```

Section 3 Reference Sheet

```
template <typename T>
class List {
public:
    // EFFECTS: Initializes this list to be empty.
    List() : first(nullptr), last(nullptr) {}
    void pop_front(); // removes the first element from the list
    void pop_back(); // removes the last element from the list
    int first_out_of_order(); // *You will implement this in part (3a)*
    void remove_all_out_of_order(); // *You will implement this in part (3c)*
    // Assume the Big Three are defined as needed.

private:
    struct Node {
        T datum; // the element stored in this Node
        Node *next; // the next Node, or null if there is none
        Node *prev; // the previous Node, or null if there is none
    };
    Node *first; // first node in the list, or null if it is empty
    Node *last;  // last node in the list, or null if it is empty
    void remove_index(int index); // *You will implement this in part (3b)*
};
```